IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON INTELLIGENT SYSTEMS FOR THE INTERNET OF THINGS

Received August 9, 2017, accepted September 8, 2017, date of publication September 29, 2017, date of current version October 25, 2017.

Digital Object Identifier 10.1109/ACCESS.2017.2755259

Hypermedia APIs for the Web of Things

JAIME A. MARTINS[1], ANDRIY MAZAYEV[1], AND NOÉLIA CORREIA[1,2]
[1]Center for Electronics, Optoelectronics and Telecommunications, University of Algarve, 8005-139 Faro, Portugal
[2]Science and Technology Faculty, University of Algarve, 8005-139 Faro, Portugal

Corresponding author: Jaime A. Martins (jamartins@ualg.pt)

This work was supported by the Portuguese Science and Technology Foundation under Project UID/MULTI/00631/2013.

ABSTRACT The *Web of Things* is a new and emerging concept that defines how the Internet of Things can be connected using common Web technologies, by standardizing device interactions on upper-layer protocols. Even for devices that can only communicate using proprietary vendor technologies, upper-layer protocols can generally provide the necessary contact points for a high degree of interoperability. One of the major development issues for this new concept is creating efficient hypermedia-enriched application programming interfaces (APIs) that can map physical Things into virtual ones, exposing their properties and functionality to others. This paper does an in-depth comparison of the following six hypermedia APIs: 1) the *JSON Hypertext Application Language* from IETF; 2) the *Media Types for Hypertext Sensor Markup* from IETF; 3) the *Constrained RESTful Application Language* from IETF'; 4) the *Web Thing Model* from Evrythng; 5) the *Web of Things Specification* from W3C; and 6) the *Web Thing API* from Mozilla.

INDEX TERMS Web of Things, Internet of Things, interoperability, hypertext, API, standard, model, HAL, Web Thing, CoRAL, HSML.

I. INTRODUCTION

One of the most important and emerging concepts related to the Internet of Things (IoT) is the *Web of Things* (WoT), which aims to define how connected devices can be accessed, interconnected and operated using standard Web technologies.

The current state of Internet and IoT-related technologies applicable in IoT scenarios is extremely fragmented, at all stack levels. There are dozens of protocols in use, at the:

(a) Physical and Link layer – e. g., IEEE 802.15.4 PHY/MAC, WiFi, Bluetooth 4.0 Low Energy and Mesh, RFID/NFC, 3GPP, WiMAX, LPWAN,
(b) Network layer – e. g., Thread, 6LoWPAN, ICMP, ROLL, RPL, IPv6, IPv4, IPSec, ZigBee RF4CE/PRO/IP,
(c) Transport layer – e. g., UDP, DTLS, MLE, TCP,
(d) Application and Data layer – e. g., Google Weave, Apple's HomeKit, CoAP, LWM2M, IPSO Smart Objects, Bluetooth HDP, DDS, DPWS, MQTT, XMPP, AMQP, SNMP, UPnP, SEP 2.0, ZeroMQ, HTTP/REST, DHCP, DNS, TLS/SSL,

among many, many others.

The solutions that build upon these protocols are quite often setup as vertical silos, each with their own proprietary technology stacks, with none or ad-hoc interoperability with other silos. This issue is the main focus of the WoT: to address interoperability at a fundamental level, by deploying a horizontal application layer that fully bridges together all of these technological silos at their top level (which would correspond to the Application layer in the TCP/IP model, or the Presentation and Application layer in the OSI model).

At its core, the WoT relies on the decades of knowledge gained from the development of the World Wide Web, and aims to embrace as many Internet of Things devices as possible, by making them discoverable, accessible and interoperable. To achieve this goal, it is imperative to carefully choose inclusive data models and forward-looking device APIs. This is a very relevant issue for anyone trying to develop and implement IoT devices today.

Following this line of thought, this paper focuses on comparing six possible Hypermedia APIs, which can be used to model WoT-enabled devices. As far as we know, no comparison has yet been made between them. These are, the:

(a) *JSON Hypertext Application Language* [12] from IETF,
(b) *Media Types for Hypertext Sensor Markup* [13] from IETF,
(c) *Constrained RESTful Application Language* [7] from IETF,
(d) *Web Thing Model* [17] from Evrythng,
(e) *Web of Things Specification* [3], [8], [10], [11] from W3C, and
(f) *Web Thing API* [4] from Mozilla.

Section II discusses each of the APIs individually, followed by a comparison in Section III. A discussion on the issues raised is presented in Section IV.

II. HYPERMEDIA APIs

A. JSON HYPERTEXT APPLICATION LANGUAGE – IETF

The JSON Hypertext Application Language (HAL) was initially created by Mike Kelly, in 2011. The most recent version of the specification is now an Internet-Draft proposal, of the IETF Network Working Group, named *draft-kelly-json-hal-08* [12].

HAL is a generic Hypermedia Web API, built to easily represent the resources of different URI endpoints and their relations. It defines a minimal amount of structure necessary for supporting a subset of hypermedia controls: *hyperlinks* (i. e., links).

Collections of arbitrary links and items are represented inside HAL Documents. For this purpose, HAL defines a new media type (**application/hal+json**). Each document is composed of a Resource Object at its root, which represents all the resources of an endpoint. The specification reserves two optional properties, **_links** and **_embedded**:

- The **_links** object (or array of objects) defines the links, and their relation types, to other endpoints [14].
- The **_embedded** object can be used to represent the contents of other (linked to) endpoints, when needed. This is very useful for caching and reducing future server requests, as embedded resources can be read directly by clients instead of having them traverse links.

```
RETRIEVE /light/ accept=application/hal+json

Response Payload:
{
  "_links": {
    "self": { "href": "/light" },
    "item": { "href": "/light/brightness" }
  },
  "_embedded": {
    "brightness": {
      "_links": {
        "collection": { "href": "/light" },
        "self": { "href": "/light/brightness" },
        "action": {"href": "/light/brightness/change"}
      },
      "brightness-percentage": 100
    }
  },
  "power-status": "on",
  "uptime-seconds": 120
}
```

Listing 1. Example of a light resource in HAL.

An example of this specification can be seen in Listing 1, for a simple light bulb. It has three properties, a powered state and an uptime counter at the **/light/** URI and a brightness percentage value at the **/light/brightness/** URI.

The usage of the **item** and **collection** link relation-types is a pattern defined in [1] and is not part of the specification, but can easily be used together with HAL, as exemplified.

B. MEDIA TYPES FOR HYPERTEXT SENSOR MARKUP – IETF

The Media Types for Hypertext Sensor Markup (HSML) [13] is an Experimental Internet-Draft, first published by the IETF Thing-to-Thing Research Group in July 2016.

HSML aims to develop and extend REST and Hypermedia design styles for machine interactions, by focusing on standardising data and interaction models of resources, as a way to increase interoperability. It is fundamentally based on previous IETF standards (most notably on Web Linking [14], CoRE Link Format [15], and SenML [9]).

Data is structured around a reusable collection pattern (serialised in JSON), which contains either *links* or *items*. New media types and hypermedia keywords are defined to interact with any collection data, allowing clients to perform state transfer operations on resources. The proposal is also compatible with CoRE Link Format and SenML, as it reuses keyword identifiers and element structures from these standards. This allows for HSML collections to be converted in CoRE Link Format and SenML representations.

The interaction model is optimised for machine workflow: (a) hyperlinks and forms are machine-comprehensible, (b) links can be embedded and/or transcluded, and (c) resource data can be combined or separated from hypertext.

HSML is also abstract from the transport layer, by generalising forms and other message-based controls, and enabling REST and pub/sub protocol bindings.

The specification defines and details the possible CRUD operations on the different resource types (i. e., collections, or just links and/or items), as well as link extensions for defining hypermedia controls on links (i. e., actions and monitors/link-bindings).

Listing 2 shows the simple light example in HSML collection format:

1) The base identifier (**bi**) of the HSML collection is set to **/light/**, which will be used for all forward link resolution.
2) An **anchor** to **/light/** is used to define the link's self-relation types. In this case, **/light/** is defined as a **collection** of items.
3) Two links (**href**) to items are shown, **power-status** and **uptime-seconds**, as well as their values at the end of the document: each **href** is directly matched to an **n** (name), which has a value (**v**). String values use the SenML keyword **vs**.
4) The **brightness** link, which is also an **item** of the **/light/** collection, has a resource type (**rt**) of **l.brightness**.
5) There is also a link-extension to **/light/brightness/scale-intensity/**, with a relation of **action**. This action has a **type** of **actuator:scaleIntensity**, and is activated using a CREATE (POST) on the link-target, with a body in the format **text/plain**, and a payload that follows the json-schema defined in the field **schema**.

```
RETRIEVE /light/ accept=application/hsml+json

Response Payload:
[
  {
    "bi": "/light/"
  },
  {
    "anchor": "/light/",
    "rel": ["self", "collection"]
  },
  {
    "href": "power-status",
    "rel": "item"
  },
  {
    "href": "uptime-seconds",
    "rel": "item"
  },
  {
    "href": "brightness",
    "rel": "item",
    "rt": "l.brightness"
  },
  {
    "anchor": "brightness"
    "href": "scale-intensity",
    "rel": "action",
    "type": "actuator:scaleIntensity",
    "method": "create",
    "accept": "text/plain",
    "schema": {
      "type": "integer", "minimum": 0, "maximum": 100 }
  },
  {
    "n": "power-status",
    "vs": "on"
  },
  {
    "n": "uptime-seconds",
    "v": 120
  },
  {
    "n": "brightness",
    "v": 100
  }
]
```

Listing 2. Example of a light resource in HSML.

C. CONSTRAINED RESTFUL APPLICATION LANGUAGE – IETF

The Constrained RESTful Application Language (CoRAL) [7] is an Experimental Internet-Draft, first published by the IETF Thing-to-Thing Research Group in March 2016.

CoRAL is a compact, binary representation format for RESTful hypermedia-driven applications, that run under constrained nodes and networks. It uses a compact serialisation format, the Concise Binary Object Representation (CBOR) [2], for defining Web links and forms in a structure that aligns closely with the Constrained Application Protocol (CoAP) [16], and is based on HAL (Section II-A).

The main CoRAL structure is called a "CoRAL document", which consists of several elements (i. e., links, forms, literals and bases). Each element type is represented by a number, a hyperlink reference, options and a body, if needed. CoRAL also supports Profiles, for defining application-specific link and form relation types. These are encoded as negative numbers.

```
RETRIEVE /light/ accept=application/coral

Response Payload (verbose):
[ [ 4,                          / base
    1,                          / absolute-path
    [ 10, "light" ] ],          /  href.path = "light"
  [ 5,                          / fat-link
    0,                          / append-relation
    [  1, 12,                   /  relation = self
       1, 32 ] ],               /  relation = collection
  [ 5,                          / fat-link
    2,                          / append-path
    [ 13, "Simple light" ] ], /  Title = "Simple light"
  [ 5,                          / fat-link
    2,                          / append-path
    [ 10, "power-status"        /  href.path = ...
       1, 33,                   /  relation = item
       4, 0 ],                  /  format = text/plain
    h'6f6e' ],                  /   "on"
  [ 5,                          / fat-link
    2,                          / append-path
    [ 10, "uptime-seconds"      /  href.path = ...
       1, 33,                   /  relation = item
       4, 9 ],                  /  format =  uint8
    h'78' ],                    /   120
  [ 5,                          / fat-link
    2,                          / append-path
    [ 10, "brightness"          /  href.path = "brightness"
       1, 33,                   /  relation = item
       16, "l.brightness",      /  rt = "l.brightness"
       4, 7 ],                  /  format = uint8
    h'64' ],                    /   100
  [ 5,                          / fat-link
    2,                          / append-path
    [ 18, "brightness"          /  anchor = "brightness"
      10, "scale-intensity",    /  href.path = ...
       1, 25,                   /  relation = action
       1,-110,                  /  relation = scaleIntensity
       4, 0,                    /  format = text/plain
       14, true ],              /  updatable = true
     h'2274797065223a2022       /   "type": "integer",
     696e7465676572222c20       /   "minimum": 0,
     226d696e696d756d223a       /   "maximum": 100
     20302c20226d6178696d       /
     756d223a20313030' ] ]      /
```

Listing 3. Example of a light resource in CoRAL.

Listing 3 exemplifies the simple light resource represented in CoRAL. The structure is very similar to HSML:

1) A base URI is defined in **/light/**, so that the remaining links of the document are relative to it.
2) Next, **/light/** is also defined as having a **self** and **collection** link relation types, followed by a **Title** of "Simple light".
3) A new link, **/light/power-status/** is defined, with a relation type of **item** and a string content (text/plain) equal to **on**.
4) The **/light/uptime-seconds/** link also has a relation type of **item** and an **uint32** format, equal to the number **120**.

5) The **/light/brightness/** link is similar, but has a resource type (rt) and uses an **uint8** format, with the number **100** as body.
6) The **/light/brightness/scale-intensity/** link has a relation type of **action** and **scaleIntensity** (custom-defined, to specify a particular type of actuator), as well as an **updatable = true**, meaning support for UPDATE (PUT) operations. The body is a hexadecimal-encoded JSON-schema of the accepted payload.

D. WEB THING MODEL – EVRYTHNG

Evrythng's Web Thing Model is based on the European project COMPOSE [5], and on the book "Building the Web of Things" [6]. The initial model was proposed as a W3C Member Submission in 2015, and the current draft (dated 25th April 2017) is accessible at [17].

This is one of the most comprehensive proposals available, and among the first to focus on the Application Layer of the Internet of Things, as a crucial point for device interoperability. The proposal conceptualises physical objects as their virtual counterparts, which are modelled into *Web Things* using several defined constructs. These are categorised by different functionality degrees, and the proposal defines requirements that must be met to achieve each category (i. e., for a *Web Thing*, an *Extended Web Thing* or a *Semantic Web Thing*).

At an architectural level, the specification defines three Integration Patterns:

(a) *Direct connectivity*, which has clients directly communicating with Web Things.
(b) *Gateway-based connectivity*, when a Web Thing cannot offer direct connectivity. A gateway can expose the Web API indirectly, as an intermediate Thing.
(c) *Cloud-based connectivity*, similar to (b), where the gateway is a remote cloud service.

Any Web Thing is able to offer resources (each with their own endpoint URI), which constitute the Web Thing Model. These are:

(a) *A Root*, as each Web Thing has a unique root resource URI.
(b) *Properties*, as variables of a Web Thing, representing internal states. These can be subscribed to, so that clients can receive notifications when specific conditions are met.
(c) *Actions*, as functions offered by a Web Thing. These can be invoked to initiate state changes.
(d) *A Model*, as an Extended Web Thing can also serve a full JSON model of itself to clients.

The specification details the JSON payload, REST API bindings and content-formats to interact with any of these resources.

An example model of an Extended Web Thing is presented in Listing 4, which can be retrieved at the **/light/model/** URI, and describes a device similar to Listing 1. In this

```
RETRIEVE /light/model/ accept=application/json

Response Payload:
{
  "id": "http://localhost:8989/light",
  "name": "Kitchen ceiling light",
  "description": "A simple lightbulb located at the
       kitchen ceiling",
  "tags": [ "light", "ceiling", "kitchen" ],
  "customFields": {
    "hostname": "localhost",
    "port": 8989,
    "path": "/light"
  },
  "links": {
    "product": {
      "link": "http://manufacturer.com/products/s-light",
      "title": "The product this Web Thing is based on"
    },
    "properties": {
      "link": "/properties",
      "title": "List of Properties",
      "resources": {
        "power-status": {
          "name": "Power Status",
          "description": "The on/off switch",
          "values": {
            "power-status": {
              "name": "Switch",
              "description": "The state of the switch",
              "unit": "on/off",
              "customFields": {
                "gpio": 12
              }
            }
          },
          "tags": [ "switch", "light" ]
        },
        "uptime-seconds": {
          "name": "Light Uptime",
          "description": "A counter of light uptime",
          "values": {
            "uptime-seconds": {
              "name": "Uptime",
              "description": "The seconds elapsed",
              "unit": "second",
              "customFields": {
                "gpio": 13
              }
            }
          },
          "tags": [ "switch", "light" ]
        },
        "brightness": {
          "name": "Light Brightness",
          "description": "Brightness of the light",
          "values": {
            "brightness-percentage": {
              "name": "Brightness",
              "description": "Percentage of Brightness",
              "unit": "percent",
              "customFields": {
                "gpio": 14
              }
            }
          },
          "tags": [ "scale", "light" ]
        }
      }
    },
```

Listing 4. Example of a light resource as an Extended Web Thing.

```
    "actions": {
      "link": "/actions",
      "title": "Actions of this Web Thing",
      "resources": {
        "brightness-change": {
          "name": "Change Brightness",
          "description": "Change the brightness of the
              Light",
          "values": {
            "brightness": {
              "type": "integer",
              "minimum": 0,
              "maximum": 100,
              "required": true
            }
          }
        }
      }
    },
    "type": {
      "link": "http://model.manufacturer.com/light-v3",
      "title": "Instance type of this Light"
    },
    "help": {
      "link": "http://manufacturer.com/docs/simple-light",
      "title": "Documentation"
    },
    "ui": {
      "link": "/",
      "title": "User Interface"
    }
  }
}
```

Listing 4. Example of a light resource as an Extended Web Thing.

case, every property and action of the light has its own URI, with the possibility of specifying data schemas, by using JSON Schema [18]. The URIs are also defined according to functionality (i. e., properties and actions are separate). The customFields keys are multi-purpose, so they are sometimes used to specify the physical locations (GPIO pin numbers) of sensors or actuators.

E. WEB OF THINGS SPECIFICATION – W3C

The W3C WoT Interest Group on the Web of Things has produced several draft proposals, since its inception at the start of 2015. These are currently under revision and consideration for standardisation, by the WoT Working Group, which was formed at the end of 2016. The key proposals are:

- The WoT Architecture [11],
- The WoT Thing Description [10],
- The WoT Scripting API [8],
- The WoT Binding Templates [3].

Each of these will be discussed below.

1) WoT ARCHITECTURE

There are three major requirements for the realisation of a functional WoT architecture [11]:

(a) *Flexibility*, so that the architectural best practices can be used in as many situations as possible, which is a complex goal when taking into consideration the wide variety of physical devices.

(b) *Compatibility*, which requires architectures to provide bridges between legacy IoT solutions and WoT implementations, as well as interoperability with current Web standards.

(c) *Security and Privacy*, as IoT/WoT solutions must have a strong emphasis on being tamper-proof, as they represent real-world objects which can be used for nefarious aims.

The functional virtual device of a physical Thing is called a "WoT Servient", which provides all access and control functions. It is composed of several key parts:

(a) *Thing Description* – Each WoT Servient relies on a *Thing Description* specification to operate, which represents the functionality of a Thing in a machine-readable format, using semantic annotation.

(b) *Scripting API* – This part specifies the APIs for interactions with Things, in 3 categories
   - A Client API for consuming Things;
   - A Server API for exposing Things;
   - A Discovery API for discovering Things.

   Security options are also defined for the 3 APIs, to deal with authentication, authorisation and secure communications.

(c) *Binding Templates* – The specification aims to define binding templates to different transfer protocols, e. g., REST-based such as HTTP and CoAP, pub-sub such as MQTT and raw channel-based such as WebSockets.

(d) *Security and Privacy* – As a cross-cutting concern, all security and privacy best practices are defined over the Thing Description and Scripting API (as both transport and object security), as well as Protocol Bindings (depending on the specific protocols).

(e) *Hardware Access* – There are two possible cases: (i) When the WoT Servient is running inside local hardware, it will access the Thing's data and functions through proprietary APIs, such as the W3C Device API or specific vendor APIs. (ii) When the WoT Servient is running outside the Thing, it must use any available legacy IoT protocols of the device, usually through proprietary vendor APIs. This implementation is defined as the Legacy Communication block of the WoT Servient.

(f) *Device Logic* – Any device logic is implemented using App Scripts, which interface with the WoT Scripting API and proprietary device APIs to access the Thing.

There are several possible Deployment Scenarios (i. e., Integration Patterns) for WoT Servients:

(a) *Direct connectivity* – When using direct communication and discovery, the WoT Server exposes its API directly to a WoT Client, through the WoT Servient. Either REST or the Client API is used to access the Server, through an IP-based network.

(b) *Gateway-based connectivity* – The WoT Servient runs on a local gateway, discovering and interfacing with local devices, while forwarding requests to/from WoT Clients. Legacy Communication can be used if needed.

The Servient exposes both the Server API (for outside WoT Client access) and the Client API (for interacting with local devices).

(c) *Cloud-based connectivity* – The WoT Servient is running as a cloud service:
- For devices able to be directly connected to the Web, the pattern is very similar to (b).
- For legacy devices, the cloud WoT Servient must interface with one or more local WoT Servient(s), running in gateway(s) with local device access. All will expose the Server and Client APIs.

The specification also exemplifies how each of these patterns can adapt the WoT Servient to a particular use case.

2) WoT THING DESCRIPTION

Each WoT Thing must be described in a specific document, named a Thing Description (TD) [10]. This document is essential for the communication between WoT Servients, and must be retrieved for each and any device. It details all available semantics and interactions of a Thing. The TD can be provided directly by each Thing, or hosted externally (e. g., in a repository) due to specific Thing restrictions (e. g., legacy devices or absence of internal storage).

The TD document is currently serialised as JSON-LD, and provides support for:

(a) *Semantic metadata*, based on an underlying RDF data model. The usage of semantic annotation facilitates the extension and integration of TDs with external contexts. Existing vocabularies can be reused and semantically combined, enhancing interoperability through semantics. Different communication bindings can also be specified (e. g., HTTP(S), CoAP), as well as media types (e. g., application/json), and security policies (e. g., authentication, authorisation).

(b) A *functional description* of a Thing's WoT Interface. A minimal vocabulary is defined, supporting three different interaction patterns:
- *Properties*, which provide readable and/or writeable, static or dynamic, data.
- *Actions*, which represent non-immediate changes or internal processes of a Thing.
- *Events*, which are able to raise notifications when certain conditions are met.

Repositories of TDs can also provide semantic query functionality (e. g., SPARQL), to retrieve the most relevant TDs.

3) WoT SCRIPTING API

This is a low-level programming specification [8] for the implementation of the constructs defined in the above subsections. It details the implementation of the WoT Runtime (part of the WoT Servient): (a) The WoT Runtime (WR) is defined as an isolated, event-loop based, script execution environment. (b) It interprets and manages the lifecycle of all WoT application scripts, and employs lower-level APIs to provide access to resources (both local and remote). (c) It serves as a security enforcement point for controlling device access.

The low-level APIs (used by the WR), are:
- A *Client API*, that allows scripts running on a Thing to discover and access other Things.
- A *Server API* for providing resources (properties, actions and events).
- A *Physical API* for accessing locally attached hardware.

4) WoT BINDING TEMPLATES

At the current time, this specification is still unreleased [3].

Listing 5 shows a possible TD for the simple light example. It is semantically enriched with **actuator** and **units** vocabularies. This allows for correspondence between arbitrary keywords and meaningful semantic terms. These are described in the linked vocabularies defined in the **@context** preamble.

F. WEB THING API – MOZILLA

One of the most recent W3C Member Submission proposals (May 2017) is the *Web Thing API*, from the Mozilla Corporation [4]. It defines a *Web Thing Description* format, in a similar way to the W3C WoT Thing Description [10] and Evrythng's Web Thing Model [17], with a minimum vocabulary for describing physical devices connected to the Web. This Description is serialised in a machine readable format (JSON), and sets the structure for specifying properties, actions and events.

The proposal also defines a REST API, which is again similar to the Web Thing Model described in Section II-D. Nevertheless, it is the first one to explicitly describe, in its Protocol Bindings, a WebSocket API for interaction with Things.

Listing 6 exemplifies a Web Thing Description for a simple light, divided into four sections: **properties**, **actions**, **events** and **links**:

1) The **properties** section has three items: (a) **power-status** which is a boolean **onOffSwitch** type (defined in Sec 5.2 of the proposal), (b) **uptime-seconds** which is a seconds counter, and (c) **brightness** which is a percentage value.
2) It is also defined a **brightness-change action** and **event**, however the proposal has yet to define the full schemas for these.
3) The links to each of the above sections are defined in the **links** section.

III. COMPARISON

Table 1 presents a comparison between the different proposals, on thirteen key areas. The table lists all specifications by first publication date, as well as the reference documents of each. The main topics of comparison are:

1) The main purpose of each one, stating the intent that lead to its creation. This is very helpful in separating specifications directly aimed at representing Things vs. the more generic ones, able to describe any kind of resource combination.

```
RETRIEVE /light/ accept=application/ld+json

Response Payload:
{
  "@context": [
    "http://w3c.github.io/wot/w3c-wot-td-context.jsonld",
    { "actuator": "http://example.org/actuator#light" },
    { "units": "http://purl.oclc.org/NET/ssnx/qu/qu-rec20#
        unit" } ],
  "@type": "Thing",
  "name": "Simple light",
  "interactions": [
  {
    "@type": [ "Property" ],
    "name": "power-status",
    "outputData": { "valueType": {
        "type": { "enum": [ "on", "off" ] }
      } },
    "writable": true,
    "links": [ {
        "href": "http://localhost:8989/light",
        "mediaType": "application/json"
      } ] },
  {
    "@type": [ "Property" ],
    "name": "uptime-seconds",
    "units:unit": "units:Seconds",
    "outputData": { "valueType": { "type": "integer" } },
    "writable": false,
    "links": [ {
        "href": "http://localhost:8989/light",
        "mediaType": "application/json"
      } ] },
  {
    "@type": [ "Property" ],
    "name": "brightness-percentage",
    "units:unit": "units:Percentage",
    "outputData": {
      "valueType": {
        "type": "integer",
        "minimum": 0,
        "maximum": 100
      } },
    "writable": false,
    "links": [ {
        "href": "http://localhost:8989/light/brightness",
        "mediaType": "application/json"
      } ] },
  {
    "@type": [ "Action", "actuator:scaleIntensity" ],
    "name": "brightness-change",
    "units:unit": "units:Percentage",
    "inputData": {
      "valueType": {
        "type": "integer",
        "minimum": 0,
        "maximum": 100 },
      "actuator:unit": "actuator:percentage"
    },
    "links": [ {
        "href": "http://localhost:8989/light/brightness/
            change",
        "mediaType": "application/json"
      } ] }
  ]
}
```

Listing 5. Example of a light resource as a W3C Thing Description.

2) The representational model. A *Thing Model* designates the specific resources of a virtual thing, and how it relates to a physical thing. Every proposal has a different way of representing these resources, and how to model the different physical (or virtual) building blocks of a Thing. These are generically represented as Properties, Actions and Events.
3) The definition of a high-level architecture specification. This item evaluates *Integration Patterns* (also called *Deployment Scenarios*): these define how Things can be integrated with the Web. Some Things can have a direct access, while others will have and indirect one, through other devices (e. g., a gateway). These patterns are fundamental to define the contexts where each specification can be used, from a connectivity and interoperability perspective. This is an important item for guiding device deployment in real-world scenarios. Standard integration patterns can help avoiding many issues regarding ad-hoc deployments.
4) A low-level API definition. This can help in structuring device operability and foster the development of software modules or plugins to extend device functionality.

```
RETRIEVE /light/ accept=application/json

Response Payload:
{
  "name": "Kitchen ceiling light",
  "type": "thing",
  "description": "A simple lightbulb located at the
      kitchen ceiling",
  "properties": {
    "power-status": {
      "type": "onOffSwitch",
      "unit": "boolean",
      "description": "The on/off switch",
      "href": "/light/power-status"
    },
    "uptime-seconds": {
      "type": "number",
      "unit": "second",
      "description": "A counter of light uptime",
      "href": "/light/uptime-seconds"
    },
    "brightness": {
      "type": "number",
      "unit": "percent",
      "description": "Brightness of the light",
      "href": "/light/brightness"
    }
  },
  "actions": {
    "brightness-change": { "description": "Change the
        brightness of the light" }
  },
  "events": {
    "brightness-change": { "description": "The brightness
        of the light was changed" }
  },
  "links": {
    "properties": "/light/properties",
    "actions": "/light/actions",
    "events": "/light/events",
    "websocket": "wss://localhost:8989/light"
  }
}
```

Listing 6. Example of a light resource in Web Thing Description.

TABLE 1. Web of things hypermedia APIs comparison.

	JSON HAL (IETF)	HSML (IETF)	CoRAL (IETF)	Web Thing Model (Evrythng)	Web of Things Specification (W3C)	Web Thing API (Mozilla)
Date of first proposal	2011	July 2016	March 2016	2014	2015	May 2017
Reference document	https://tools.ietf.org/html/draft-kelly-json-hal-08	https://datatracker.ietf.org/doc/draft-koster-t2trg-hsml	https://datatracker.ietf.org/doc/draft-hartke-t2trg-coral	http://model.webofthings.io	https://w3c.github.io/wot-thing-description	http://iot.mozilla.org/wot
Main purpose?	Defining a new media type for representing resources and relations, using hyperlinks	Defining a media type and CRUD for exploration and interaction, as a hypermedia API	Compact, binary representation format for building RESTful, hypermedia-driven applications that run in constrained environments	Defining Web Things	Defining Web Things	Defining Web Things
What does it model?	Collections of arbitrary links (many) and items (many), by resource	Collections of arbitrary links (many) and items (only one), by resource	Collections of arbitrary links (many) and items (many), by resource. The data model is derived from HAL	Thing models and data	Thing interaction models/templates	Thing interaction models and templates
High-level Architecture definition? (i. e., Itegration Patterns)	No	No	No	Yes	Yes	Yes
Definition of a low-level Scripting API and Runtime Environment?	No	No	No	No	Yes, WoT Scripting API and WoT Runtime. Defines explicitly: (i) a Server API, (ii) a Client API, and (iii) a Physical API	No
Defines interaction bindings for different protocols?	No	Yes	Yes	Yes	Yes	Yes
Has a specific model for defining/templating Things?	No, implicit definition	No, implicit definition	No, implicit definition	Yes, explicit, as an *Extended Web Thing*	Yes, explicit, as *Thing Descriptions*	Yes, explicit, as *Thing Descriptions*
Clearly separates data models and templates from instanced data?	Partially. Arbitrary definition of the link meaning (with rel). Instanced data from links is defined under the _embedded sub-object	Arguably, yes. Templates are defined in links and link-extensions and instanced data is defined in items. Does not currently define instances of link-extensions	Similar to HSML	Yes, the Thing's model can be found on the rel = model link	Yes, whole TDs can be interexchanged or be on a remote repository	Yes, the Thing's model can be found on the root endpoint
Defines the structure of interaction content (e.g., actions) besides templating?	N/A. Does not define Forms for interaction	No	No	Yes	No	Yes
Defines semantic extensions?	No	No	No	Yes, optional, using JSON-LD, for *Semantic Web Things*	Yes, required, using JSON-LD	Yes, optional, using JSON-LD
Defines Security and Privacy guidelines?	No	Yes, light guidelines, optional implementation	Not yet. Section "*7. Security Considerations*" is currently empty	Yes, light guidelines, optional implementation	Yes, explicitly, but as an optional implementation	No
Relies on non-standard media types?	Yes, application/hal+json	Several, with application/hsml+json as primary	Yes, application/coral	No, uses standard application/json or application/ld+json	No, uses standard application/ld+json for TDs	No, uses standard application/json or application/ld+json
Principal advantages	Aims for generality and simplicity	**(a)** Clean media-type, allowing for both human- and machine-readability. **(b)** Integrates the CoRE Link Format and SenML in the same media-type	**(a)** Has the most compact representation of all, even without CBOR. **(b)** Can transparently encode CoRE Link Format. **(c)** Can serialize any arbitrary item data representation, including JSON objects and RDF	**(a)** Currently the most thought-out model for Things. **(b)** Explicitly defines the content of Form interaction instances	**(a)** Huge interest, the WoT IG has currently 214 active members. **(b)** Currently the spec undergoing most development. **(c)** Clearly defines Things and their usage patterns	**(a)** Defines WebSocket bindings. **(b)** Tries to join concepts from the Web Thing Model and the W3C WoT spec
Potential drawbacks	**(a)** Does not specify hypermedia Forms for interaction. **(b)** Has no concept of a Thing	**(a)** Very recent. **(b)** The work of just one author. **(c)** Will probably suffer restructuring for a unified T2T RG proposal. **(d)** Has no concept of a Thing. **(e)** The new media type does not follow the structure of either Core Link Format or SenML data, requiring new parsers and rules for transcoding	**(a)** Very recent. **(b)** The work of just one author. **(c)** Will probably suffer restructuring for a unified T2T RG proposal. **(d)** Has no concept of a Thing	Has no clear backing from a standards body	The Protocol Bindings specification [3] is still empty	**(a)** The most recent. **(b)** Protocol Bindings spec in very early stage. **(c)** Integration patterns spec in very early stage

5) Protocol bindings. These define how each specification behaves when interacted by different types of protocols.
6) The concept of a cohesive Thing model. This reflects the constraints and recommendations of each specification on how to model and represent a Thing and its resources. Some specifications choose to group resources cohesively, under a higher-level entity (i. e., a Thing), while others leave that choice open.
7) Model and instancing separation. Is there a way to retrieve the complete model of a device, or is the model intermixed with data, leaving the separation responsibility to the client?
8) Operationalisation of interactions. Is there a data-model for non-immediate interactions (e. g., like activating device actions), or is the structure of the information to be shown left to each particular implementation?
9) Semantic support. This is an important field to take into consideration when designing Things which will need to be related semantically to other Things or resources.
10) Security and privacy guidelines. A very crucial field for Things which will have direct Internet access, and whose interactions can create physical danger.
11) Proprietary content format. Standards-based approaches are generally preferred to proprietary formats, as they increase interoperability. However, some formats are adequate for very specific purposes or integration patters (e. g., like solutions for very constrained devices).
12) The principal advantages, which summarises the strengths of each model.
13) The potential drawbacks, which summarises the weak points of each model, which are to be considered depending on the needs of each implementation scenario.

IV. Discussion

This manuscript presented several state-of-the-art proposals for the Web of Things, which is a research field experiencing significant growth during the last few years. There is a clear tendency between standards bodies to address this problem in two different ways, using either a:

(I) Bottom-up approach – where both related and unrelated individual units of data (e. g., items) and functionality (e. g., hyperlinks) are collected inside documents (i. e., collections), which can represent (or not) individual Things (e. g., like different atoms which must be analysed and related, to construct a cohesive molecule).
(II) Top-down approach – where Things are envisioned as a cohesive unit of data and functionality (e. g., like a molecule, which is comprised of different atoms).

The bottom-up approach is clearly used by HAL (Section II-A), HSML (Section II-B) and CoRAL (Section II-C). The top-down approach is used by Evrythng's Web Thing Model (Section II-D), W3C-IG's Web of Things (Section II-E) and Mozilla's Web Thing API (Section II-F).

The bottom-up approach has its strengths on being able to represent any type of complex Web resources, not just Things. However, it is up to the client to discover what belongs to who, and responsible to create itself a rough ''Thing model'', to interact with a pool of related resources. However, there is never a guarantee that it is looking at the whole picture (as information can be scattered in various places), and must be able to reason with incomplete information.

The top-down approach has its strengths on presenting cohesive concepts of Things: a Model or TD can be sent as a whole to any client, and the relations between different components are explicit (and possibly, semantically linked). The strengths of this approach are also its drawbacks: representing resources not directly Thing-related, or combining pieces of different Things, is anti-pattern: a client expects that everything presented in a model belongs together, cohesively.

Most of the presented specifications are also very young. Some of them will probably be replaced or obsoleted in time. CoRAL and HSML are both being developed by single authors of the IETF's Thing-to-Thing Research Group, and are very probable candidates for being combined into a single specification. HAL is still lacking hypermedia forms 6 years later, so it will hardly get a more universal traction in the WoT space.

Mozilla's Web Thing API also follows very closely Evrythng's Web Thing Model, which has been around since 2014 and is currently one of the most thorough specifications. However, it lacks the backing of a standards body or a big industry name, like Mozilla, who could accelerate the dissemination of a revised version of this proposal.

Among the most well positioned proposals are the W3C WoT specifications, which are clearly advancing the WoT standardisation in a transversal way, from the high-level architecture definition to the low-level scripting APIs. However, both Mozilla and Evrythng's proposals are also W3C Member Submissions. This leaves to the W3C WoT Working Group the responsibility of choosing the best proposals, or even combining the strengths of each. In the end, their final proposal could be very different from any of the ones currently under consideration.

ACKNOWLEDGEMENTS

This work was developed within the Centre for Electronic, Optoelectronic and Telecommunications (CEOT).

REFERENCES

[1] M. Amundsen, *The Item and Collection Link Relations*, document RFC 6573, 2012.
[2] C. Bormann and P. Hoffman, *Concise Binary Object Representation (CBOR)*, document RFC 7049, 2013.
[3] U. Davuluru and M. Kovatsch. (2017). ''Web of things (WoT) binding templates,'' Tech. Rep. [Online]. Available: https://w3c.github.io/wot-binding-templates/
[4] B. Francis. (2007). ''Web thing API, W3C member submissions,'' Tech. Rep. [Online]. Available: http://iot.mozilla.org/wot/

[5] D. Guinard and V. Trifa, "Towards the Web of Things: Web mashups for embedded devices," in *Proc. WWW Int. World Wide Web Conf. Work. Mashups, Enterp. Mashups Light. Compos. Web (MEM)*, Madrid, Spain, 2009, pp. 1–8.
[6] D. Guinard and V. Trifa, *Web Things*. Greenwich, CT, USA: Manning Publications Co., 2015.
[7] K. Hartke, *The Constrained RESTful Application Language (CoRAL)*, document Internet-Draft draft-hartke-t2trg-coral-02, Internet Engineering Task Force, 2017. [Online]. Available: https://datatracker.ietf.org/doc/draft-hartke-t2trg-coral/
[8] J. Hund, Z. Kis, and K. Nimura. (2017). "Web of things (WoT) scripting API," Tech. Rep. [Online]. Available: https://w3c.github.io/wot-scripting-api/
[9] C. Jennings, Z. Shelby, J. Arkko, A. Keranen, and C. Bormann, *Media Types for Sensor Measurement Lists (SenML)*, document IETF Stand. Track Internet-Draft, (draft-ietf-core-senml-10), 2017. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-core-senml/
[10] S. Kaebisch and T. Kamiya. (2017). "Web of things (WoT) thing description," Tech. Rep. [Online]. Available: https://w3c.github.io/wot-thing-description/
[11] K. Kajimoto, U. Davuluru, R. Matsukura, J. Hund, M. Kovatsch, and K. Nimura. (2017). "Web of Things (WoT) architecture," Tech. Rep. [Online]. Available: https://w3c.github.io/wot-architecture/
[12] M. Kelly, *JSON Hypertext Application Language*, document IETF Informational Internet-Draft, (draft-kelly-json-hal-08), 2016. [Online]. Available: https://datatracker.ietf.org/doc/draft-kelly-json-hal/
[13] M. Koster, *Media Types for Hypertext Sensor Markup*, document Internet-Draft draft-koster-t2trg-hsml-01, Internet Engineering Task Force, 2017. [Online]. Available: https://datatracker.ietf.org/doc/draft-koster-t2trg-hsml/
[14] M. Nottingham, *Web Linking*, document RFC 5988, 2010.
[15] Z. Shelby, *Constrained RESTful Environments (CoRE) Link Format*, document RFC 6690, IETF, 2012.
[16] Z. Shelby, K. Hartke, and C. Bormann, *The Constrained Application Protocol (CoAP)*, document RFC 7252, IETF, 2014.
[17] V. Trifa, D. Guinard, and D. Carrera. (2017). "Web thing model, W3C Member Submission," Tech. Rep. [Online]. Available: http://model.webofthings.io/
[18] K. Zyp and G. Court, *JSON Schema: Core Definitions and Terminology*, document Internet Engineering Task Force, 2013. [Online]. Available: https://datatracker.ietf.org/doc/draft-zyp-json-schema/

JAIME A. MARTINS received the degree in electronics and telecommunications engineering (5 years) and in clinical psychology (4 years) from the University of Algarve in 2001 and 2007, respectively, and the Ph.D. degree in computer science (in 3-D face and object recognition) from the University of Algarve in 2014. He is currently a full-time Researcher with the Networks and Systems Group, Center for Electronics, Optoelectronics and Telecommunications, a research center supported by the Portuguese Foundation for Science and Technology. His major interests span several emerging IT areas, from the Internet of Things to computer/human vision, and cognitive neuroscience.

ANDRIY MAZAYEV received the B.Sc. and M.Sc. degrees in computer science from the University of Algarve, Faro, Portugal, in 2012 and 2015, where he is currently pursuing the Ph.D. degree in wireless sensor networks and a Collaborator of the Networks and Systems Group, Center for Electronics, Optoelectronics and Telecommunications. His major research interests include the Internet of Things, semantic Web, Web protocols, ubiquitous computation, and wireless sensor networks in general.

NOÉLIA CORREIA received the B.Sc. and M.Sc. degrees in computer science from the University of Algarve, in Faro, Portugal, in 1995 and 1998, respectively, and the Ph.D. degree in optical networks (computer science) from the University of Algarve in 2005, and done in collaboration with University College London, U.K.

She is a Lecturer with the Science and Technology Faculty, University of Algarve, in Faro, Portugal. Her research interests include the application of optimization techniques to several network design problems, in the optical, wireless, and sensor networks fields, and development of algorithms.

She is a Founding Member of the Center for Electronics, Optoelectronics and Telecommunications, University of Algarve, a research center supported by the Portuguese Foundation for Science and Technology. She is also the Networks and Systems Group Co-ordinator.